3Н

УДК 004.93'1;004.932

А.М. Недзьведь, П.В. Лукашевич, А.М. Белоцерковский
Объединённый институт проблем информатики Национальной академии наук Беларуси
Беларусь, 220012, г. Минск, ул. Сурганова, 6

Гибкая система обработки изображений на базе скрипт-ядра с использованием интеллектуального агента

A.M. Nedzved, P.V. Lukashevich, A.M. Belotserkovsky
United Institute of Informatics Problems of The National Academy of Sciences, Belarus
Belarus, 220012, Minsk, Surganova str., 6.

A Flexible Image Processing System Based on a Script-Kernel Using an Intelligent Agent

А.М. Недзьведь, П.В. Лукашевич, А.М. Білоцерківський
Об'єднаний інститут проблем інформатики Національної академії наук Білорусі
Білорусь, 220012, м. Мінськ, вул. Сурганова, 6

Гнучка система обробки зображень на базі скрипт-ядра з використанням інтелектуального агента

В статье предлагаются концептуальные схемы управления процессами в системе обработки изображений, позволяющие создавать графический интерфейс в соответствии с прикладной спецификой решаемой задачи. Схемы позволяют изменять и наращивать функциональность системы, так как построены на скрипт-ядре с использованием «интеллектуального» коллаборативного подхода, что дает возможность применять объективные и проверенные алгоритмы обработки изображений.
Ключевые слова: скриптовой язык, графический интерфейс пользователя, автоматизированная обработка изображений, управление процессами.

Conceptual schemes for process management of image processing system is proposed here so to generate a Graphic User Interface, according to specific needs of certain users and developers. Schemes allow modifications and building-up the functionality since it based on a script-kernel and intelligent support by collaborating approach, which gives ability to use objective and reliable algorithms to solve applied tasks.
Key words: scripting language; GUI; automated image analysis; process management.

У статті пропонуються концептуальні схеми управління процесами в системі обробки зображень, що дозволяють створювати графічний інтерфейс відповідно до прикладної специфіки користувачів і розробників. Схеми дозволяють змінювати і нарощувати функціональність системи, оскільки побудовані на скрипт-ядрі з використанням «інтелектуального» колаборативного підходу, що дає можливість для розв'язання прикладних завдань застосовувати об'єктивні і перевірені алгоритми обробки зображень.
Ключові слова: скриптова мова, графічний інтерфейс користувача, автоматизована обробка зображень, управління процесами.

Введение

Даже если принять во внимание только область обработки изображений, в настоящее время существует большое количество программных библиотек, коммерческих или разработанных под лицензией «Open Source». Как правило, поддержка таких

инструментов – далеко не тривиальная работа, диктующая определенную специфику в организации программного обеспечения. Она зависит от решаемой задачи, опыта пользователя, а также среды пользовательского рабочего места. По этой причине, современные требования к программному обеспечению заключаются в необходимости динамичной организации функционала, пользовательского интерфейса и управления данными в данной предметной области (методы обработки изображений, контроль качества, дистанционный контроль и т.п.).

Любое программное приложение из области анализа изображений «исследует» объект в терминах его происхождения, структуры и функциональных свойств (например, в медицинских задачах [1]). Достаточно сложно спроектировать компьютерную реплику инструментов для автоматизированного анализа изображений с тем, чтобы создать универсальное рабочее место. Таким образом, необходимо рассмотреть возможность использования технологических решений, в основе которых лежит так называемая «адаптивность» или «гибкость» ПО («Software Flexibility»).

В настоящее время «гибкость» ПО (не путать с гибкой методологией разработки, англ.: «Agile software development») – достаточно популярная концепция, как новое свойство программных приложений. Определение было впервые дано IEEE [2]: «Легкость, с которой система или компонент могут быть модифицированы для использования в приложениях или в условиях других, чем те, для которых она была специально разработана» (перевод авторов). Определение, безусловно, может быть оспорено или, по крайней мере, дополнено по ряду позиций, тем не менее, существует целый ряд метрик, количественно и качественно описывающих степень гибкости разрабатываемой программной среды [3]. Обсуждение этих вопросов выходит за рамки данной статьи, однако авторы преследуют глобальную цель предложить новое архитектурное решение прикладной предметно-ориентированной системы, которая бы в дальнейшем позволила:

– улучшенный анализ изображений путем получения численных, повторимых и объективных результатов;

– выбор релевантных подмножеств признаков в диагностических задачах путем отслеживания множеств признаков, наиболее используемых экспертами;

– получение новых знаний при выделении новых признаков и исполнение статистических экспериментов для оценки их эффективности;

– новые возможности в программной поддержке принятия решений.

На данном этапе предлагается элегантная концепция управления процессами в системе обработки изображений и создания пользовательского интерфейса с применением интеллектуальной социальной (рекомендательной) составляющей, что позволяет вносить изменения в функционал, надстраивать его с помощью скриптов обработки и упростить решение описанных выше проблем.

Скрипт как основа гибкого ПО

В прикладной программе сценарий (скрипт) – это программа, которая автоматизирует ту или иную задачу, которую без сценария пользователь делал бы вручную, используя интерфейс программы [4], [5]. Скриптовые языки позволяют разработчикам комбинировать и «сцеплять» вместе различные функции или пакеты программ, а также согласовывать результаты, полученные в результате работы системы в целом. Для создания гибкой среды или программной системы с «открытой архитектурой» вышеописанные языки являются эффективными с точки зрения их интеграции как инструмента модификации и расширения функционала разрабатываемого ПО для анализа изображений.

При выборе оптимального скриптового языка, но котором должно основываться ядро программного комплекса с «открытой архитектурой», был использован материал, опубликованный Гийомом Марсо [6]. В своем исследовании он сравнил параметры 72 реализаций языков программирования по 19 специальным тестам, подготовленным проектом «The Computer Language Benchmarks Game». Оценивались такие параметры, как скорость исполнения кода и его размер, потребление памяти, необходимой для реализации определенных функций и т.д. Гибкость и особый функционал языка Lua [7] позволил определить его как наиболее подходящий при решении поставленной задачи. Благодаря использованию интерпретатора Lua в качестве ядра, система обрела следующие достоинства:

1. Гибкость интеграции отдельных задач в едином операционном пространстве. Реализация различных компонент инвариантна к процессу их ассемблирования и адаптации под конкретные задачи.

2. Декомпозиция задачи на ряд небольших подзадач, с учетом системности, т.е. более эффективное распределение заданий между членами групп и разработчиками, упрощение процесса отладки.

3. Простая портируемость без изменений и рекомпиляции уже созданных и отлаженных компонент в новые проекты, и их интеграция в новую систему связей, схему функционирования и информационного обмена с другими компонентами.

4. Возможность использования отлаженного сценария сборки в качестве основы для нового проекта с новым или доработанным функционалом.

5. Проектируемость системы по принципу «от простого к сложному», постепенно наращивая функциональные возможности системы и решая первоочередные задачи проекта.

Структура разрабатываемого программного обеспечения с учетом особенностей задачи организована так, что базовый язык большинства функций C++ может одновременно расширять Lua. Например, С-функции могут вызывать Lua-функции и наоборот. Основой симбиотического взаимодействия между Lua и его базовым языком является виртуальный стек, который является структурой данных LIFO («Last In-First Out», «последний вошел – первый вышел») и временно сохраняет аргументы функции и ее результаты. Для вызова из Lua базового языка (и наоборот) вызывающая сторона помещает значения в стек и вызывает целевую функцию; принимающая сторона извлекает аргументы из стека, обрабатывает данные и помещает в стек результаты. Когда управление возвращается вызывающей стороне, она извлекает значения из стека. Используя C API, приложение может также получить информацию из Lua-структуры. Обратное действие (вызов С-функции из Lua) аналогично. Lua может загружать и вызывать функции по требованию.

В результате интеграции Lua в программу появилась возможность писать и экспортировать в Lua функции программы, написанные на С, а также использовать эти функции прямо из скрипта программы. Кроме того, подобные функции можно создавать и использовать как отдельные специализированные DLL модули (с Lua интерфейсом). В силу большой распространенности языка существует большое количество уже разработанных модулей, также есть возможность при помощи специально разработанных функций сопряжения использовать функции обычных dll-библиотек.

Использование описанной методологии для гибкого программного обеспечения позволяет получить возможность быстрой адаптации интерфейса для различных прикладных задач, в частности, и эффективное управление логикой и функционалом системы, в целом. Как следствие, интерактивная составляющая уменьшается, что существенно упрощает работу в той или иной области.

Создание гибкого пользовательского интерфейса

Графический интерфейс пользователя (Graphic User Interface, GUI) или ГИП приложения состоит из двух наборов функций: неизменяемых базовых статичных функций и расширяемого набора, который управляется посредством Lua-скриптов. Сложные элементы GUI программируются на С++ как Lua-функция в виде единого компонента и к ним относятся такие формы, как «виджет» (контейнер) отображения и редактирования изображения с соответствующими инструментами, диалог интерактивной пороговой сегментации, диалог отображения гистограммы изображения, таблицы результатов вычисления характеристик и текстовый редактор работы с LUA скриптом. Простые элементы напрямую транслируются в Lua-функции и к ним относятся: управление меню, панели с управляющими элементами и универсальный диалог, который включает элементы, необходимые для вызова Lua-функций (рис. 1).

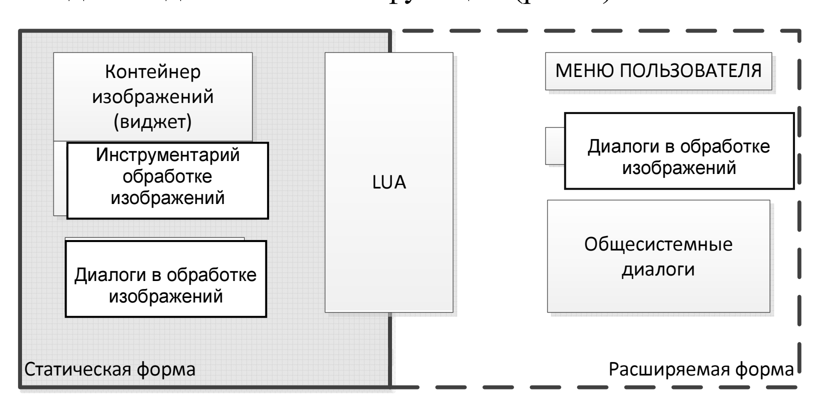


Рисунок 1 – Графический интерфейс пользователя «гибкого» приложения

Работа с приложением, сознанным по предлагаемой структуре, сводится к следующим основным принципам:

– GUI управляется через LUA, который является корневым элементом интерфейса программы – контейнером, где размещаются все элементы;

– позиционирование элементов в контейнере производится при помощи GUI. Он сам расположит и отобразит элемент согласно определенным правилам. Основные контейнеры: фрейм, вертикальный сайзер, горизонтальный сайзер;

– обработчики событий задаются в виде функций, «прикрепленных» к «виджетам»;

– цикл обработки событий запускается после создания диалога;

– универсальный диалог покрывает все необходимые элементы для вызова функций LUA.

Таким образом, достигается простое управление внешним видом приложения и быстрая адаптация интерфейса программы к разнородным задачам анализа изображения. При такой организации приложения интерактивная часть работы может быть сведена к минимуму, что значительно упрощает взаимодействие пользователя с программой.

Технология описанная в данной статье предназначена для организации управления процессами с тем, чтобы:

– адаптировать интерфейс под нужды конкретного специалиста (пользователя);

– использовать доступный алгоритмический опыт пользователей;

– модифицировать алгоритмы в соответствии со спецификой той или иной задачи и предоставить адекватный динамический интерфейс взаимодействия с пользователем;

– оптимизировать решение задач запуском процессов в распределенном режиме под управлением веб-сервиса.

Очевидно, что все описанные вопросы связаны с особой архитектурой построения графического интерфейса, но не только. В проектируемой системе должен также присутствовать некоторый Агент, который будет принимать решение о том, какие именно изменения должны быть внесены в интерфейс и функционал.

Рекомендательная система и коллаборативная фильтрация

Предлагаемая в статье система имеет свойства, схожие со свойствами рекомендательной системы, в основе которой лежит следующее предположение: те пользователи, которые дали оценку тем или иным объектам какого-либо типа в прошлом, могут дать похожее оценку другим схожим объектам и в будущем. Решения принимаются индивидуально каждым пользователем, при этом используемая информация извлекается из базы данных. Наиболее распространённым подходом для решения задачи предсказания (прогнозирования) в рекомендательных системах является коллаборативная фильтрация (Collaborating Filtering), использующая известные предпочтения и оценки группы пользователей, чтобы сделать предположение о неизвестных предпочтениях нового пользователя [8].

Коллаборативная фильтрация отличается от простого подхода, так как присваивает некоторое число (голосов) исследуемому объекту, например, в результате голосования. В настоящее время проводятся достаточно активные исследования в этой области, поэтому существует ряд методов, решающих задачу с использованием различного математического аппарата.

В проектируемой программе предлагается три режима работы рекомендательной системы, используемых при различном объеме информации о предпочтениях конкретного пользователя:

– информация отсутствует (так называемый «холодный старт», например, сразу после регистрации пользователя);

– набрана информация об интенсивности использования определенного набора функций (стратегия пассивной обратной связи с пользователем, никаких действий от пользователя не требуется; качество рекомендаций растет с интенсивностью использования);

– помимо информации об интенсивности использования, также набрана база количественных оценок используемых функций (стратегия активной обратной связи с пользователем, от пользователя требуется вручную выставлять оценки используемым алгоритмам по пятибалльной шкале; качество рекомендаций существенно растет с ростом базы оценок; стратегия оптимальна для быстрого старта).

Итак, для сценария 1 при «холодном старте» предлагается для персонификации интерфейса использовать либо усреднённую оценку алгоритма всеми пользователями, либо усреднённую оценку алгоритма пользователями в данной сфере знаний (при ее наличии) или различных типов изображений, например, спутниковые изображения, медицинские КТ, гистологические или цитологические изображения, наноскопические структуры и т.д.

В дальнейшем добавление социальной составляющей и использование отзывов других пользователей позволит максимально быстро формировать такой «слабо изменяемый костяк ГИП» на этапе «холодного старта» прямо после регистрации, основываясь лишь на сфере деятельности пользователя и его профиле.

Для сценария 2 используется подход на базе решений коллаборативной фильтРации по предметам на основании статистики покупок, посещений, голосов. Для реали-

зации был выбран алгоритм «Item-to-Item» [9]. По сравнению с принятым методом, в систему предлагается внести модификацию: за факт использования сценария принималось не однократное его исполнение, а применение того же сценария не менее 10 раз.

Для сценария 3 используется подход на базе решений коллаборативной фильтрации для предметов с оценками на базе хорошо зарекомендовавшего себя алгоритма «Slope One» [10].

Интеллектуальный агент для управления скриптами

При разработке архитектуры задача-ориентированной системы рекомендательного типа было сделано предположение, что существует, по крайней мере, две различные схемы создания интерфейса с интеллектуальной поддержкой, функционирующие в соответствии с первым сценарием (без фильтрации) и вторым или третьим (с фильтрацией). Встроенные интеллектуальные агенты (ИА) авторы предлагают условно называть соответственно: «ИА без фильтрации» и «ИА с фильтрацией».

По сути, основа ИА в предлагаемых схемах одна и та же. Разница заключается лишь в том, что в первом случае генерация GUI-скрипта выполняется по предустановленному набору оценок скриптов обработки, основанных на выделенных признаках изображения. В другом случае оценки формируются адаптивно в зависимости от количества областей пользовательских интересов (в данном случае в области анализа изображений). Таким образом, качество оценки увеличивается с увеличением числа пользователей или интенсивности использования программы, если система запущена локально для одного пользователя.

Интеллектуальный агент без фильтрации. Пользовательский интерфейс в этом случае генерируется с помощью Lua-скриптов на основании вычисленных характеристик обрабатываемого изображения (например: панель для текстурных характеристик, если исследуется текстура).

Так как приложение работает в полностью автономном режиме и не использует любую другую информацию кроме самого изображения, то набор рекомендаций закладывается разработчиком приложения заранее, на этапе его создания и распространения. Схема клиентской части представлена на рис. 2.

Так, например, при загрузке фотоизображения в ГИП будут актуализированы алгоритмы цветовой коррекции, улучшения качества, алгоритмы текстурного анализа и т.п., а при работе с бинарными изображениями на первый план выйдут морфологические преобразования, алгоритмы дистанционных преобразований и скелетизации, функции выделения и подсчета «блобов».

На первый взгляд, предлагаемая схема накладывает ограничения на возможности, предоставляемые пользователю, однако, благодаря скриптовой реализации Интеллектуальной составляющей, опытные пользователи могут изменять как алгоритмическую (вычислительную) функциональность, так и интерфейс.

Интеллектуальный агент с фильтрацией. ГИП в этом случае также генерируется с помощью Lua-скриптов на основе вычисленных характеристик изображения с применением ИА. Однако рекомендации по использованию определенного функционала системы делаются при помощи коллаборативной фильтрации, собирающей и обрабатывающей статистику использования определенных функций. На рис. 3 показана принципиальная схема архитектуры клиента по обработке изображений с расширяемым функционалом и встроенным интеллектуальным агентом с фильтрацией.

Таким образом, дальнейшей работой стало развитие упомянутого метода, основной сложностью которого при всех его преимуществах является создание управляющего

скрипта интеллектуального агента, который мог бы обеспечить логику адаптации графического интерфейса для заданного типа изображений. Эта задача достаточно просто и качественно решается на фиксированном наборе выделяемых характеристик изображений и фиксированном наборе скриптов обработки посредством обучения на тестовой выборке, как это описано выше. Для того, чтобы выполнить условие гибкости системы, потребовалась адаптация ИА для работы на расширенных наборах признаков и алгоритмов.

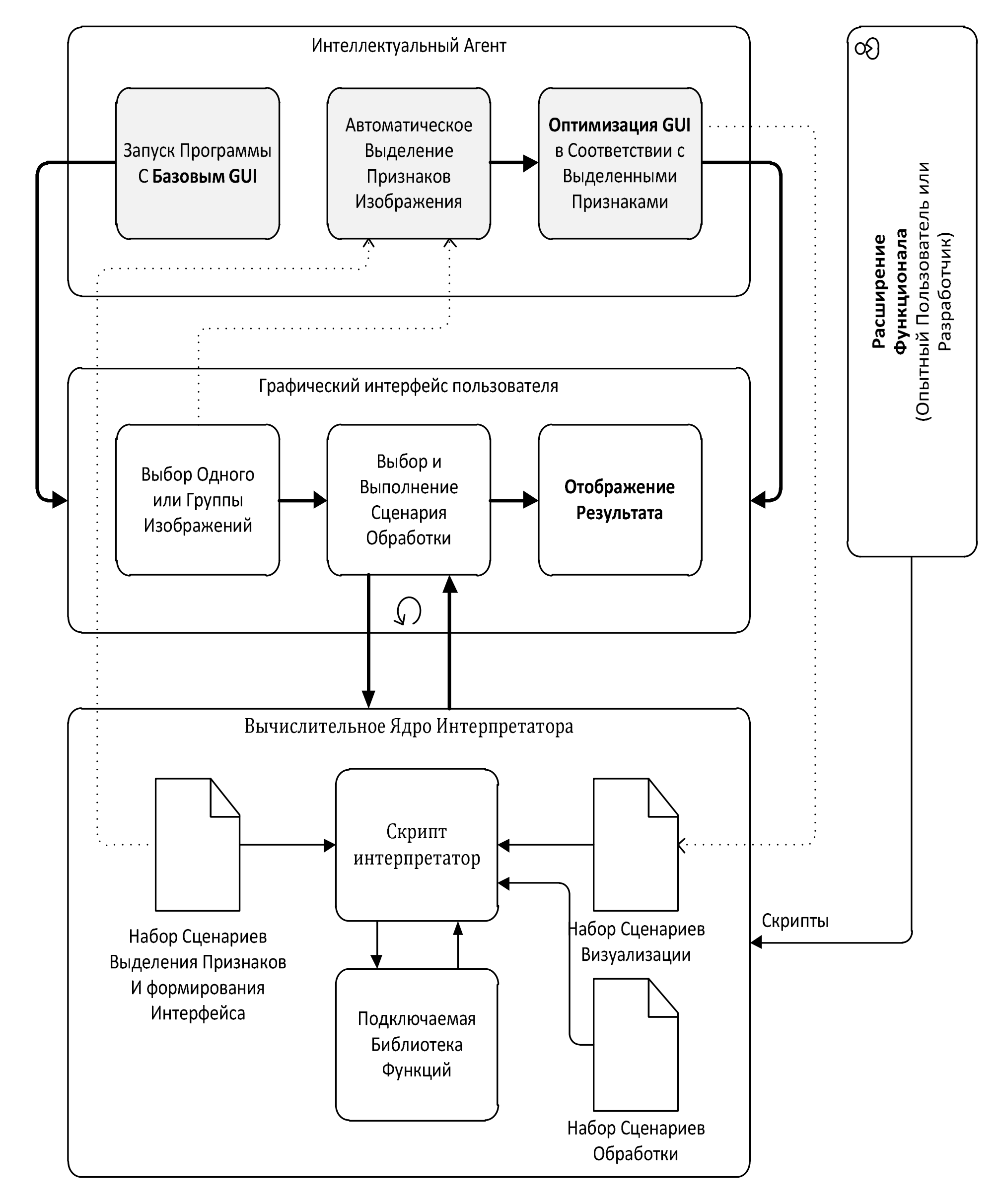


Рисунок 2 – Архитектура системы обработки изображений (клиент) с расширяемым функционалом без фильтрации

С целью решить поставленную задачу был предложен способ адаптации логики работы ИА на основании пользовательской истории использования программы. Для этого ИА в фоновом режиме производит вычисление признаков всех исследуемых изображений и фиксирует в локальной базе данных факт применения для них определенного набора алгоритмов и частоту их использования. Далее, при построении

интерфейса проводится анализ локальной БД и на основании пользовательской истории использования приложения формируется персонализированный интерфейс.

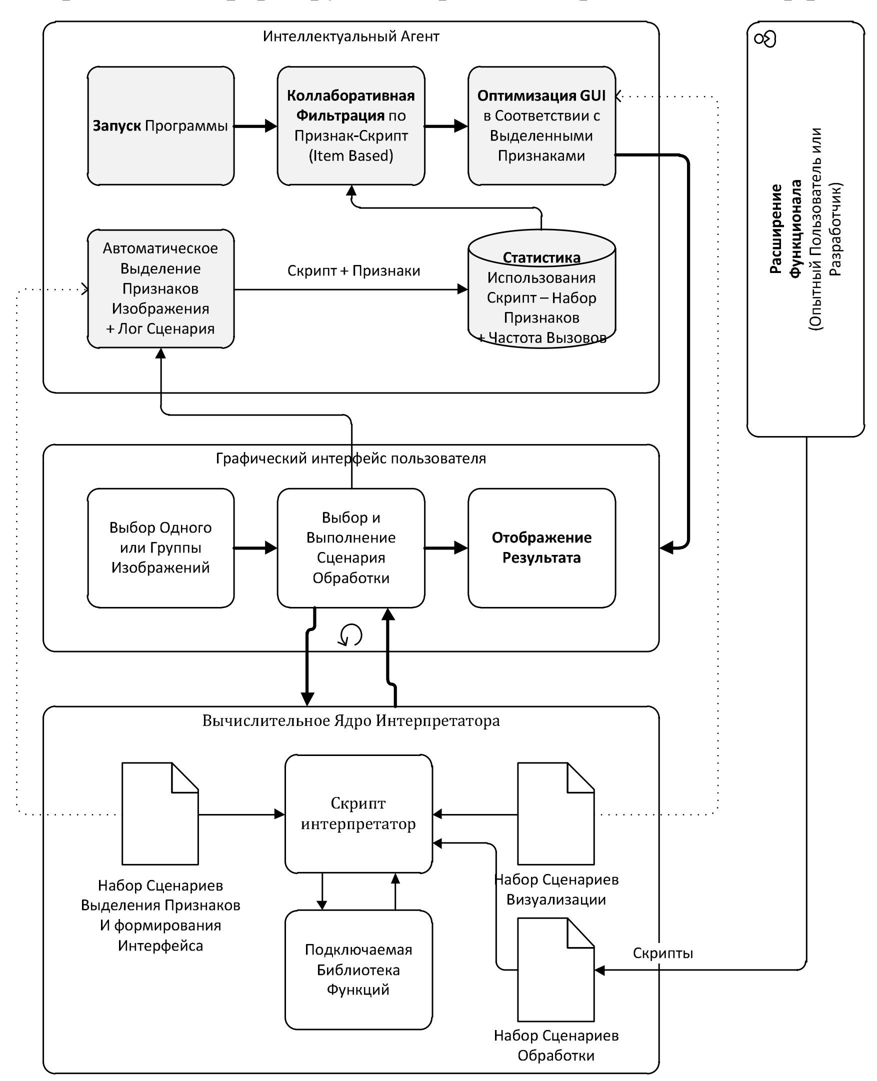


Рисунок 3 – Архитектура системы обработки изображений (клиент) с расширяемым функционалом без фильтрации

Наличие локальной базы данных и такой статистики позволяет не только рекомендовать интерфейс, уникальный для изображений с определенными свойствами, но и формировать персонифицированную основу (слабо изменяемый костяк) ГИП для этого пользователя, отражающую особенности его работы.

Заключение

В основе описанной концепции заложено использование скриптового языка как ядра системы, что позволяет сделать графический интерфейс пользователя динамичным и изменяемым. Данная технология создания ПО также дает возможность генерировать новые функции, расширять функционал, и вносить, таким образом, гибкость, делая работу с системой более удобной.

В статье описана только клиентская часть системы. Проектирование серверной части также может внести ряд полезных функциональных свойств, таких, как накопление статистики использования сценариев и эффективного применения методов коллабо-

ративной фильтрации для формирования персонализированного интерфейса пользователя на основании истории его отзывов (или статистики использования сценариев). Развивая предложенный подход, можно рассмотреть идею создания централизованного хранения (репозитория) с тем, чтобы применить простой механизм для обновления базы алгоритмов обработки изображений.

Централизованное хранение базы алгоритмов обработки изображений имеет ряд преимуществ, среди которых:

– возможность использования актуальных сценариев обработки изображений;
– возможность оперативного исправления/добавления сценариев;
– минимальные затраты при расширении функционала программы;
– возможность добавления новых сценариев без перекомпиляции разработчиками и опытными пользователями прямо из интерфейса клиентской части программы.

Еще один шаг развития концепции – создание и использование множеств новых программируемых функций, которые могут быть сконфигурированы с помощью Lua-скриптов под управлением Интеллектуального агента.

Литература

1. Nedzvedz A. Self-Developing Software for Image Processing Tasks / A. Nedzvedz, A. Belotserkovsky, I. Gurevich [and other] // Proc. On Open German-Russian Workshop «Pattern Recognition and Image Understanding». – Nizhny Novgorod, 2011. – November 21 –26. – P. 215-218.
2. IEEE. Standard Glossary of Software Engineering Terminology 610.12-1990, Vol. 1. Los Alamitos: IEEE Press, 1999.
3. Amnon H. Eden. Measuring Software Flexibility / Amnon H. Eden, Tom Mens // IEE Software. – London, UK : The Institution of Engineering and Technology, 2006, Jun. – Vol. 153, № 3. – P. 113-126.
4. Bard Bloom. Robust Scripting via Patterns / Bard Bloom, Martin Hirzel // Proc. On Dynamic Languages Symposium (DLS). – 2012. – P. 29-40.
5. Ierusalimschy R. Programming with multiple paradigms in lua / R Ierusalimschy // Functional and Constraint Logic Programming Lecture Notes in Computer Science. – 2010. – Volume 5979. – P. 1-12.
6. Marceau G. The speed, size and dependability of programming languages // Blog «Square root of x divided by zero», 2009. – http://gmarceau.qc.ca/blog/2009/05/speed-size-and-dependability-of.html.
7. Roberto Ierusalimschy. Passing a Language through the Eye of a Needle / Roberto Ierusalimschy, Luiz Henrique de Figueiredo, and Waldemar Celes. – 2011. – Queue 9 (5). – P. 20-30.
8. Xiaoyuan Su. A Survey of Collaborative Filtering Techniques / Xiaoyuan Su, Taghi M. Khoshgoftaar // Advances in Artificial Intelligence. – 2009. – P. 19.
9. Greg Linden. Amazon.com Recommendations: Item-to-Item Collaborative Filtering / Greg Linden, Brent Smith, Jeremy York // IEEE Internet Computing. – 2003. – Jan/Feb. – Vol. 07, № 1. – P. 76-80.
10. Daniel Lemire. Slope One Predictors for Online Rating-Based Collaborative Filtering / Daniel Lemire, Anna Maclachlan // In SIAM Data Mining (SDM’05). – Newport Beach, California, April 21-23, 2005.

RESUME

A.M. Nedzved, P.V. Lukashevich, A.M. Belotserkovsky

A Flexible Image Processing System Based on a Script-Kernel Using an Intelligent Agent

In this paper the concept is proposed, which is based on inherent use of a scripting language as a core of the system with the property of flexibility. The technology of creating software also allows generating new features and extending the functionality in whole. It makes working much more convenient and increase usability of the system.

It has been shown that developed system refers to recommending type. Thus, the use of an embedded intelligent agent can generate scripts based on gathered statistics and effectively apply the techniques of collaborative filtering for generating personalized user interface based on the log of his reviews (or statistics regarding use of scripts).

Статья поступила в редакцию 12.04.2013..